Data and Preproccessing

Read the data from pdf file

In [1]:
```
#Read the pdf with PyPDF2

#!pip install PyPDF2
import PyPDF2

with open('cocacola_10k.pdf', 'rb') as file:
    reader = PyPDF2.PdfReader(file)
    num_pages = len(reader.pages)
    print(num_pages)
    page = reader.pages[0]
    text = page.extract_text()
    print(text)
```

```
183
UNITED ST ATES
SECURITIES AND EXCHANGE COMMISSION
WASHINGTON, D.C. 20549
FORM 10-K
(Mark One)
☒ ANNUAL  REPOR T PURSUANT  TO SECTION 13 OR 15(d) OF  THE SECURITIES EXCHANGE ACT  O
F 1934
For the fiscal year  ended December  31, 2022
OR
☐ TRANSITION REPOR T PURSUANT  TO SECTION 13 OR 15(d) OF  THE SECURITIES EXCHANGE ACT
OF 1934
For the transition period fr om to
Commission File Number  001-02217
COCA COLA CO
(Exact name of Registrant as specified in its charter)
Delaware 58-0628465
(State or other jurisdiction of incorporation) (I.R.S. Employer Identification No.)
One Coca-Cola Plaza
Atlanta,Georgia 30313
(Address of principal executive offices) (Zip Code)
Registrant’s telephone number, including area code: (404) 676-2121
Securities registered pursuant to Section 12(b) of the Act:
Title of each class Trading Symbol(s) Name of each exchange on which registered
Common Stock, $0.25 Par Value KO New York Stock Exchange
0.500% Notes Due 2024 KO24 New York Stock Exchange
1.875% Notes Due 2026 KO26 New York Stock Exchange
0.750% Notes Due 2026 KO26C New York Stock Exchange
1.125% Notes Due 2027 KO27 New York Stock Exchange
0.125% Notes Due 2029 KO29A New York Stock Exchange
0.125% Notes Due 2029 KO29B New York Stock Exchange
0.400% Notes Due 2030 KO30B New York Stock Exchange
1.250% Notes Due 2031 KO31 New York Stock Exchange
0.375% Notes Due 2033 KO33 New York Stock Exchange
0.500% Notes Due 2033 KO33A New York Stock Exchange
1.625% Notes Due 2035 KO35 New York Stock Exchange
1.100% Notes Due 2036 KO36 New York Stock Exchange
0.950% Notes Due 2036 KO36A New York Stock Exchange
0.800% Notes Due 2040 KO40B New York Stock Exchange
1.000% Notes Due 2041 KO41 New York Stock Exchange
Securities registered pursuant to Section 12(g) of the Act: None
```

Extract the Management’s Discussion and Analysis of Financial Condition and Results of Operations session from the PDF file

In [2]:

```python
#Extract MDA Section

import re

mda_start = re.compile(r"Management’s Discussion and Analysis of Financial Condition a
mda_end = re.compile(r"Quantitative and Qualitative Disclosures About Market Risk")

mda_section = ""
for line in text.split('\n'):
    if mda_start.search(line):
        mda_section += line + '\n'
    elif mda_end.search(line):
        break
```

```
    else:
        mda_section += line + '\n'
```

Preprocess the extracted text by lowerring the case, removing punctuations, stop words and carring out stemming and lemmatization to normalize the text

In [3]:
```
#preprocess text
#convert to lowercase, reove punctuation, remove stopwords
#stemming and lemmatization

from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string

#lowercase
mda_section = mda_section.lower()

# remove punctuation
mda_section = mda_section.translate(str.maketrans("", "", string.punctuation))

#tokenize and remove stopwords
stop_words = set (stopwords.words('english'))
tokens = word_tokenize(mda_section)
filtered_tokens = [token for token  in tokens if token not in stop_words]

#stemming and lemmatization
ps = PorterStemmer()
lemmatized_tokens = [ps.stem(token) for token in filtered_tokens]
```

In [4]:
```
#view the lemmatized tokens
lemmatized_tokens
```

Out[4]:
```
['unit',
 'st',
 'ate',
 'secur',
 'exchang',
 'commiss',
 'washington',
 'dc',
 '20549',
 'form',
 '10k',
 'mark',
 'one',
 '☒',
 'annual',
 'repor',
 'pursuant',
 'section',
 '13',
 '15d',
 'secur',
 'exchang',
 'act',
 '1934',
 'fiscal',
 'year',
 'end',
 'decemb',
 '31',
 '2022',
 '□',
 'transit',
 'repor',
 'pursuant',
 'section',
 '13',
 '15d',
 'secur',
 'exchang',
 'act',
 '1934',
 'transit',
 'period',
 'fr',
 'om',
 'commiss',
 'file',
 'number',
 '00102217',
 'coca',
 'cola',
 'co',
 'exact',
 'name',
 'registr',
 'specifi',
 'charter',
 'delawar',
 '580628465',
 'state',
```

```
'jurisdict',
'incorpor',
'ir',
'employ',
'identif',
'one',
'cocacola',
'plaza',
'atlantageorgia',
'30313',
'address',
'princip',
'execut',
'offic',
'zip',
'code',
'registr',
'',
'telephon',
'number',
'includ',
'area',
'code',
'404',
'6762121',
'secur',
'regist',
'pursuant',
'section',
'12b',
'act',
'titl',
'class',
'trade',
'symbol',
'name',
'exchang',
'regist',
'common',
'stock',
'025',
'par',
'valu',
'ko',
'new',
'york',
'stock',
'exchang',
'0500',
'note',
'due',
'2024',
'ko24',
'new',
'york',
'stock',
'exchang',
'1875',
'note',
'due',
```

```
                '2026',
                'ko26',
                'new',
                'york',
                'stock',
                'exchang',
                '0750',
                'note',
                'due',
                '2026',
                'ko26c',
                'new',
                'york',
                'stock',
                'exchang',
                '1125',
                'note',
                'due',
                '2027',
                'ko27',
                'new',
                'york',
                'stock',
                'exchang',
                '0125',
                'note',
                'due',
                '2029',
                'ko29a',
                'new',
                'york',
                'stock',
                'exchang',
                '0125',
                'note',
                'due',
                '2029',
                'ko29b',
                'new',
                'york',
                'stock',
                'exchang',
                '0400',
                'note',
                'due',
                '2030',
                'ko30b',
                'new',
                'york',
                'stock',
                'exchang',
                '1250',
                'note',
                'due',
                '2031',
                'ko31',
                'new',
                'york',
                'stock',
                'exchang',
```

```
'0375',
'note',
'due',
'2033',
'ko33',
'new',
'york',
'stock',
'exchang',
'0500',
'note',
'due',
'2033',
'ko33a',
'new',
'york',
'stock',
'exchang',
'1625',
'note',
'due',
'2035',
'ko35',
'new',
'york',
'stock',
'exchang',
'1100',
'note',
'due',
'2036',
'ko36',
'new',
'york',
'stock',
'exchang',
'0950',
'note',
'due',
'2036',
'ko36a',
'new',
'york',
'stock',
'exchang',
'0800',
'note',
'due',
'2040',
'ko40b',
'new',
'york',
'stock',
'exchang',
'1000',
'note',
'due',
'2041',
'ko41',
'new',
```

```
 'york',
 'stock',
 'exchang',
 'secur',
 'regist',
 'pursuant',
 'section',
 '12g',
 'act',
 'none']
```

Sentiment Analysis

In [5]:
```
#Sentiment Analysis

#using textblob
#!pip install textblob
from textblob import TextBlob


text = ''.join(lemmatized_tokens)
mda_bolb = TextBlob(text)
sentiment = mda_bolb.sentiment.polarity
print("Sentiment: ", sentiment)
```

```
Sentiment:  0.0
```

In [6]:
```
#using nlkt for the sentiment analysis

from nltk.sentiment.vader import SentimentIntensityAnalyzer

sia=SentimentIntensityAnalyzer()

text = ''.join(lemmatized_tokens)
sentiment = sia.polarity_scores(text)
print("Sentiment: ", sentiment['compound'])
```

```
Sentiment:  0.0
```

The sentiment analysis is 0 which means the text is neutral

Historical Stock Price Analysis

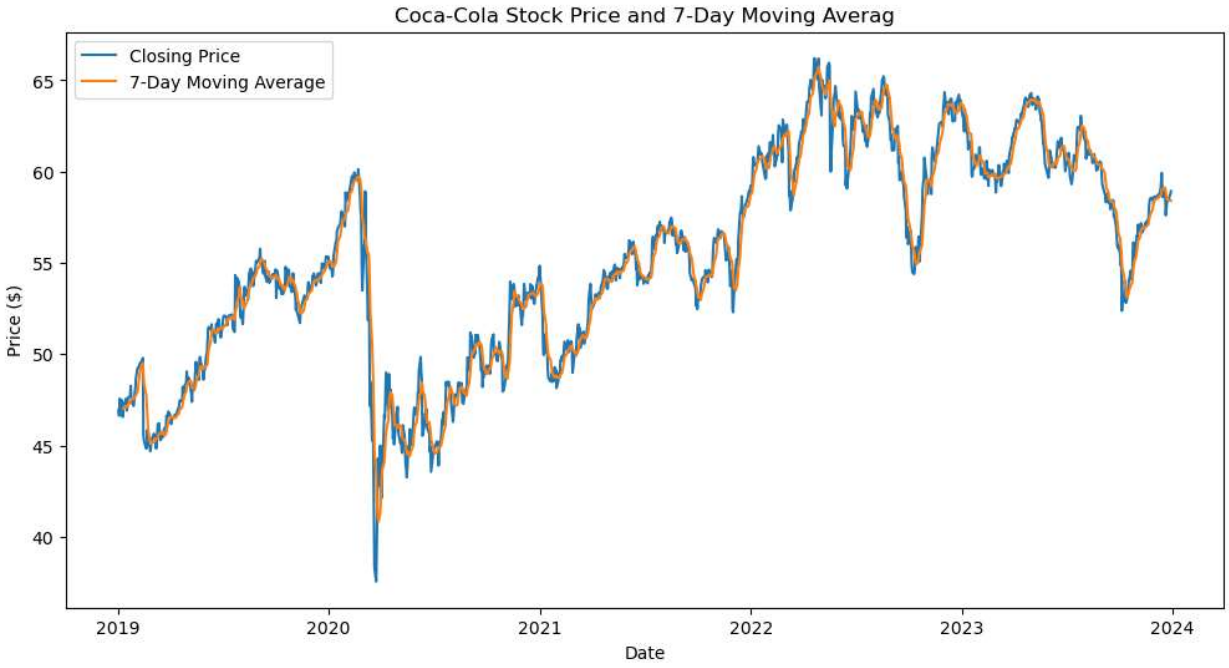
In [7]:
```
# getting historical data of stock price from yfinance

#!pip install yfinance
import yfinance as yf

data = yf.download(tickers="KO", start="2019-01-01", end="2023-12-31")

#print closing price of data
print(data['Close'])
```

```
[**********************100%%**********************]  1 of 1 completed
Date
2019-01-02    46.930000
2019-01-03    46.639999
2019-01-04    47.570000
2019-01-07    46.950001
2019-01-08    47.480000
                 ...
2023-12-22    58.320000
2023-12-26    58.560001
2023-12-27    58.709999
2023-12-28    58.750000
2023-12-29    58.930000
Name: Close, Length: 1258, dtype: float64
```

In [8]:
```
#calculating moving average price of closing price of stock to see trend of stock

data['7-Day MA'] = data['Close'].rolling(window=7).mean()
print(data['7-Day MA'])
```

```
Date
2019-01-02          NaN
2019-01-03          NaN
2019-01-04          NaN
2019-01-07          NaN
2019-01-08          NaN
                 ...
2023-12-22    58.487143
2023-12-26    58.418572
2023-12-27    58.434286
2023-12-28    58.395715
2023-12-29    58.410000
Name: 7-Day MA, Length: 1258, dtype: float64
```

In [9]:
```
#Plot the closing prices and moving average prices of stock to visualize trend of stoc

import matplotlib.pyplot as plt

plt.figure(figsize=(12,6))
plt.plot(data['Close'], label = "Closing Price")
plt.plot(data['7-Day MA'], label= "7-Day Moving Average")
plt.title("Coca-Cola Stock Price and 7-Day Moving Averag")
plt.xlabel("Date")
plt.ylabel("Price ($)")
plt.legend()
plt.show()
```



Generally, the stock has an upward trend

Feature Extraction

In [10]:
```
#Using TermFrequecy-Inverse Document Frequency (TF-IDF) to get features by converting


from sklearn.feature_extraction.text import TfidfVectorizer


#Initiling the vectorizer
vectorizer = TfidfVectorizer()

#fit vectorizer and transform the data
X = vectorizer.fit_transform([''.join(lemmatized_tokens)])

#print shape to see dimentioned of transformed data
print(X.shape)
```

```
(1, 4)
```

Machine Learning

In [11]:
```
# using the exracted features as X and sentiment as y
#the document is duplicated to create a binary classification problem

import numpy as np

X = vectorizer.fit_transform([' '.join(lemmatized_tokens), ' '.join(lemmatized_tokens)

# convert the sparse matric to a dense array
X = X.toarray()

#recall sentiment
sentiment = mda_bolb.sentiment.polarity
```

```
y = np.array([sentiment, -sentiment]) #first document is positive and second document

#split into training and test sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.5, random_state=4
```

In [12]:
```
# for Naive Bayes

from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gnb_model = GaussianNB()
gnb_model.fit(X_train, y_train)

y_pred = gnb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Naives Bayes Accuracy: ", accuracy)
```

```
Naives Bayes Accuracy:  1.0
C:\Users\USER\anaconda3\lib\site-packages\sklearn\naive_bayes.py:489: RuntimeWarning:
divide by zero encountered in log
  n_ij = -0.5 * np.sum(np.log(2.0 * np.pi * self.var_[i, :]))
C:\Users\USER\anaconda3\lib\site-packages\sklearn\naive_bayes.py:490: RuntimeWarning:
invalid value encountered in true_divide
  n_ij -= 0.5 * np.sum(((X - self.theta_[i, :]) ** 2) / (self.var_[i, :]), 1)
```

In [13]:
```
# for Support Vector Machine

from sklearn.svm import SVC

svc_model = SVC(kernel="Linear", random_state=0)
svc_model.fit(X_train, y_train)

y_pred = svc_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("SVM Accuracy: ", accuracy)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5616\1107178248.py in <module>
      4
      5 svc_model = SVC(kernel="Linear", random_state=0)
----> 6 svc_model.fit(X_train, y_train)
      7
      8 y_pred = svc_model.predict(X_test)

~\anaconda3\lib\site-packages\sklearn\svm\_base.py in fit(self, X, y, sample_weight)
    197             )
    198
--> 199         y = self._validate_targets(y)
    200
    201         sample_weight = np.asarray(

~\anaconda3\lib\site-packages\sklearn\svm\_base.py in _validate_targets(self, y)
    718         self.class_weight_ = compute_class_weight(self.class_weight, classes=
cls, y=y_)
    719         if len(cls) < 2:
--> 720             raise ValueError(
    721                 "The number of classes has to be greater than one; got %d cla
ss"
    722                 % len(cls)

ValueError: The number of classes has to be greater than one; got 1 class
```

In [14]:
```
print(np.unique(X_train))
```

```
[0.02297182 0.04594364 0.06891546 0.09188728 0.1148591  0.34457729
 0.36754911 0.39052093 0.45943639]
```

In [15]:
```
print(y_train)
```

```
[0.]
```

Since the y_class only has one unique value which is 0 due to the sentiment being neutral, the SVM would not work as it requires more than one class

In [16]:
```
# for Decision Tree

from sklearn.tree import DecisionTreeClassifier

dt_model = DecisionTreeClassifier(random_state=0)
dt_model.fit(X_train, y_train)

y_predict = dt_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Decision Tree Accuracy: ", accuracy)
```

```
Decision Tree Accuracy:  1.0
```

Both Decision Tree and Naive Bayes gives an accuracy of 1

Performance of diversified Stocks

The Performance metrics : Mean Absolute Percentage Error (MAPE) and Mean Absolute Error (MAE) are used to evaluate performance of diversified stocks.

In [17]:
```
# Calculate 7-day moving average

data['7-Day MA'] = data['Close'].rolling(window=7).mean()


# MAPE

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred)/y_true))*100

mapes = []
for i in range(len(data)):
    mapes.append(mean_absolute_percentage_error(data["Close"][:i+1], data['7-Day MA'][

# MAE
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(data["Close"], data['7-Day MA'])

print("Mean Absolute Percentage Error (MAPE): ", sum(mapes)/len(mapes))
print("Mean Absolute Error: ", mae)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_5616\2844578222.py in <module>
     16 from sklearn.metrics import mean_absolute_error
     17
---> 18 mae = mean_absolute_error(data["Close"], data['7-Day MA'])
     19
     20 print("Mean Absolute Percentage Error (MAPE): ", sum(mapes)/len(mapes))

~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py in mean_absolute_error(y
_true, y_pred, sample_weight, multioutput)
    189     0.85...
    190     """
--> 191     y_type, y_true, y_pred, multioutput = _check_reg_targets(
    192         y_true, y_pred, multioutput
    193     )

~\anaconda3\lib\site-packages\sklearn\metrics\_regression.py in _check_reg_targets(y_
true, y_pred, multioutput, dtype)
     94     check_consistent_length(y_true, y_pred)
     95     y_true = check_array(y_true, ensure_2d=False, dtype=dtype)
---> 96     y_pred = check_array(y_pred, ensure_2d=False, dtype=dtype)
     97
     98     if y_true.ndim == 1:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in check_array(array, accep
t_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow
_nd, ensure_min_samples, ensure_min_features, estimator)
    798
    799         if force_all_finite:
--> 800             _assert_all_finite(array, allow_nan=force_all_finite == "allow-na
n")
    801
    802     if ensure_min_samples > 0:

~\anaconda3\lib\site-packages\sklearn\utils\validation.py in _assert_all_finite(X, al
low_nan, msg_dtype)
    112         ):
    113             type_err = "infinity" if allow_nan else "NaN, infinity"
--> 114             raise ValueError(
    115                 msg_err.format(
    116                     type_err, msg_dtype if msg_dtype is not None else X.dtype

ValueError: Input contains NaN, infinity or a value too large for dtype('float64').
```

To avoid/ deal with the error, the presence of NaN values is checked and replaced with mean of data.

In [18]:
```
data['7-Day MA'] = data['Close'].rolling(window=7).mean()

#check for NaN values in the moverng average data
print(data['7-Day MA'].isnull().sum())
```

```
6
```

In [19]:
```
#replace NaN values with mean
data['7-Day MA'] = data['7-Day MA'].fillna(data['7-Day MA'].mean())
```

In [20]:
```
#check for NaN values in the closing price data
print(data['Close'].isnull().sum())
```

```
0
```

The presence of infinity values is also checked

In [21]:
```
#check for and handle infinity values in the moverng average data
print(np.isinf(data['7-Day MA']).sum())
```

```
0
```

In [22]:
```
#check for and handle infinity values in the closing price data
print(np.isinf(data['Close']).sum())
```

```
0
```

The MAPE and MAE is then calculated

In [23]:
```
# MAPE

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs((y_true - y_pred)/y_true))*100

mapes = []
for i in range(len(data)):
    mapes.append(mean_absolute_percentage_error(data["Close"][:i+1], data['7-Day MA'][

# MAE
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(data["Close"], data['7-Day MA'])

print("Mean Absolute Percentage Error (MAPE): ", sum(mapes)/len(mapes))
print("Mean Absolute Error: ", mae)
```

```
Mean Absolute Percentage Error (MAPE):  1.7459576571339992
Mean Absolute Error:  0.6794928356827612
```

The small MAPE indicates a small persentage error in prediction likewise the low MAE indicates a small absolute error in the predictions.

These metrics suggest that the diversified stocks strategy (since the moving average is used as a proxy for the predicted stock prices) perform well. The relatively small MAPE and MAE indicate that the predictions were close to actual stock prices.

In [ ]:
```
```